Spring 2013

BIG DATA CLASSIFICATION USING DECISION TREES ON THE CLOUD

Chinmay Bhawe
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Bhawe, Chinmay, "BIG DATA CLASSIFICATION USING DECISION TREES ON THE CLOUD" (2013). *Master's Projects*. 317.

DOI: <https://doi.org/10.31979/etd.z672-cqqn>
https://scholarworks.sjsu.edu/etd_projects/317

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

BIG DATA CLASSIFICATION USING DECISION TREES ON THE CLOUD

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of Requirements for the Degree

Master of Science

by

Chinmay Bhawe (007840040)

Spring 2013

© 2013

Chinmay Bhawe

ALL RIGHTS RESERVED

The Designated Thesis Committee Approves the Thesis Titled

BIG DATA CLASSIFICATION USING DECISION TREES ON THE CLOUD

by

Chinmay Bhawe

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2013

Dr. Chris Tseng
Department of Computer Science

Date

Dr. Tsau Young Lin
Department of Computer Science

Date

Aditya Kulkarni
Sr. Software Developer at Google Inc.

Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research

Date

Abstract

BIG DATA CLASSIFICATION USING DECISION TREES ON THE CLOUD

Chinmay Bhawe

This writing project addresses the topic of attempting to use machine learning on very large data sets on cloud servers.

The project consists of two phases. The first being developing a machine learning system which will learn on the data provided by IBM for the “IBM Watson Great minds Challenge SJSU Pilot” competition and providing the best possible results on the evaluation data set, also provided by the IBM Watson team. This will serve as a basis for the second phase of the project, in which the objective is to move the machine learning system on to a cloud server, so that it may be used as a service by future students.

The innovation in this project would be to use machine learning based data classification techniques on the cloud and solve a real world classification problem. The challenges involved would be first deploying and testing the classification algorithm that was developed in CS297 on the cloud. The project consists of not just the study of the different techniques of machine learning and its applications, but also involves identifying the algorithm and the environment which will be most suitable for this particular classification problem.

Table of Contents

1. Project Overview	8
1.1 Introduction	8
1.2 Problem Statement	8
1.3 Problem Challenges	10
2. Project Design	11
2.1 Understanding the data and preparing it	11
2.2 Using Decision Trees to solve the problem	13
2.3 Overfitting, Pruning and Cross Validation	16
2.4 Classifying an unseen instance	17
2.5 Machine Learning environments and libraries	18
3. Phase 1 – Using MATLAB for the IBM Challenge	19
3.1 Setting up and using MATLAB	19
3.2 Preview of the MATLAB code	21
3.3 How to run the code	22
3.4 Results for Phase 1	24
4. Phase 2 – Using Weka for Cloud based system	25
4.1 Why Weka?	25
4.2 Setting up the AWS Server	27
4.3 Installing and setting up Weka	29
4.4 Steps to run training and classification using Weka	29
4.5 Results for Phase 2	37
4.6 Observations and Inferences	38
4.7 MATLAB vs. Weka	41
5. Conclusion	42
6. Proposed future work	43
7. References	44

List of Table and Figures

Figure 1 : Sample instances data and Decision Tree	14
Figure 2: Preview of the MATLAB Decision Tree program code	21
Figure 3: Preview of results and run statistics	23
Figure 4: Weka Home GUI.....	26
Figure 5: Weka file pre-process example	26
Figure 6: Weka file classification example.....	27
Figure 7: AWS Launch EC2 Instance.....	28
Figure 8: AWS Choose Operating System	28
Figure 9: WinSCP File Transfer demo	30
Figure 10: Sample ".arff" file.....	31
Figure 11: J48 Tree training command example	33
Figure 12: J48 tree building statistics	34
Figure 13: Sample results file after classification.....	36
Figure 14: Table of experiments in Weka.....	38
Figure 15: Training time vs training data size	39
Figure 16: Time taken to classify vs. validation data size	40

ACKNOWLEDGEMENT

Through this acknowledgement, I would like to convey my utmost gratitude to all those who have helped me or were associated to this this project in any way. They have made this a great learning experience.

I would like to thank Dr. Chris Tseng for giving me the opportunity to undertake this project. The fruition of my efforts is owing to his guidance and his support. I would like to thank the Department of Computer Science at SJSU and all the professors at the department who I have had the pleasure of learning from or being guided by in any way. I would like to thank Dr. Lin and Mr. Aditya Kulkarni for being part of my committee, and giving me their insightful opinions and comments.

I would also like to thank my family, my friends, colleagues and anyone who has helped me with their thoughts and opinions in successfully completing this project.

1. Project Overview

1.1 Introduction

I realized the power of machine learning during my research and work in machine learning for the “Watson Great Minds Challenge” competition conducted by IBM. The aim of this contest was to attempt to apply new machine learning algorithms to the IBM Watson supercomputer learning system in order to classify a large set of answers into 2 groups- relevant or not relevant to the particular question. Machine learning techniques are very useful in pattern recognition and classification problems. Personal experience with IBM’s Watson supercomputer machine learning techniques revealed that the computer algorithms are efficient and accurate in classifying a particular group of inputs into different classes if they are allowed to “learn” on a sufficient size of classified input.

1.2 Problem Statement

The objective of the challenge is to develop an algorithm that can assign labels to the evaluation and final data sets with the highest level of accuracy possible. All project submissions will contain the question ID in the first column and the matching label that their algorithm produces in the second column.

The second part of the problem involves migrating the developed algorithm to a cloud server, and testing the performance.

In the IBM challenge, we were provided with one very large “.csv” file, consisting of about 3 million rows and 343 columns. This was a small part of the labeled data set that

is used to train IBM's supercomputer Watson on questions pertaining to the popular television game show Jeopardy. The first column was the question ID, the next 341 were feature vectors and the last column was the outcome, either true or false. This basically determined whether the particular row was relevant to the question ID (true) or not (false). The range for the values of the features was also unknown, and owing to the size of the file, extremely difficult to find out. All that could be inferred was that the 341 features were all numeric, in nature, and were signed numbers. This meant that there were positive as well as negative values. The 3 gigabyte file contained approximately 1.3 million instances.

The challenge was to build a machine learning system using any set of software libraries or machine learning tools such that it will learn based on the large data set provided, build a model, and then be able to classify an unlabeled set of Jeopardy questions (in the feature vector form) as relevant to the row id.

The machine learning system was trained and then unlabeled data of similar format was classified using the decision tree generated. The data used was from the question and answer set used to train IBM's Watson supercomputer for the "Jeopardy" game. In this process I successfully learned about the different methodologies of machine learning such as decision and classification trees, neural networks, support vector machines, etc., and also gained practical experience in working with classification on large data sets and all the challenges that come with that.

This data was very large compared to what machine learning systems are generally exposed to. As a result, many machine learning libraries and software suites were of not much use, as they were not designed to handle this sort of “big data”.

1.3 Problem Challenges

There were several challenging aspects in the course of the project .The first challenge was to understand the data set that was provided. With very little information about the data and no information provided about the very large number of features in the data or their significance, it was difficult to make sense of the data.

Another challenge was to decide on which machine learning algorithm should be chosen to tackle the problem in hand. The objective was to balance the accuracy of the prediction and the speed at which the entire data could be trained on and classified.

The file itself was so big, that no conventional file editing programs could open the file, including MS Excel 2010. Also, most freely available machine learning libraries are not designed to handle data of this magnitude. Many programs could not even ingest the whole 3 gigabytes of data, let alone train on it.

The second phase of the project provided a new challenge. I was required to develop or copy the machine learning system developed to a new open-sourced software library, which could be legally hosted on a cloud server. This involved learning a new software suite and adapting the learning algorithm to the new system. Furthermore, making sure that the accuracy and the results achieved in the second phase using the open source tools at least matches if not exceeds those achieved in phase 1 was another challenge.

2. Project Design

2.1 Understanding the data and preparing it

After reviewing various articles on machine learning, classification problems, and also trying out various different tools which provided machine learning environments, it was clear that there is no best solution to all classification problems. Every classification problem will differ in terms of the best solution in terms of the algorithm, the tool to use and the type and size of data required for learning.

Machine learning is a technique by which a computer “learns” from a set of data given to it, and is then able to predict the result of new data similar to the training data. The machine learning algorithm is meant to identify patterns based on different characteristics or “features” and then make predictions on new, unclassified data based on the patterns “learned” earlier. The input data is usually numerous instances of relations between the different variables or features relevant to the data. Usually, larger the training data set, the more accurate the prediction results are.

In the domain of machine learning, a classification problem involves the machine learning attempting to predict to which class/group/category a new observation would most likely belong to, based on the training data set, which contains already classified data. The instances are divided into a set of discrete valued set of properties, known as various features of the data. For example, classifying a received email as “spam” or “not spam” could be based on analyzing characteristics of the email such as origin ip address, the number of emails received from the same origin, the subject line, the email address

itself, the content of the body of the email, etc. All these features will contribute to a final value which will allow the algorithm to classify the email. It is logical that the more number of examples of spam and non-spam emails the machine learning system goes through, the better will be its prediction for the next unknown email.

Problems which involve classification are considered to be instances of a branch of machine learning called as “supervised learning”. In this, the machine is given a “training set” of correctly classified instances of data in the first stage, and then the algorithm devised from this “learning” is used for the next stage of prediction. The converse of this is “unsupervised learning”, which involves classifying data into categories based on some similarity of input parameters in the data.

In machine learning classification problems, a feature vector is an n -dimensional vector of numerical values which represent the different characteristics of the particular instance of the data. As an example, the features of an image instance might be the RGB color values of the individual pixels, or those of a particular text could be the occurrence frequencies of each different word in the text. Feature vectors values are often combined with weights such that it becomes easier to normalize all the value ranges and calculate a total score for classification.

The entire classification process involves a lot of intermediate steps such as data preprocessing, clustering, feature construction, feature selection, classification, regression and finally visualization. So as to reduce the size of the space in consideration, a various

different dimensionality reduction techniques are usually employed before selecting the required features.

2.2 Using Decision Trees to solve the problem

There are various different approaches to machine learning namely neural networks, decision trees, clustering, Bayesian networks, reinforcement learning, support vector machines, genetic algorithms, and many more.

I decided to use Decision and Classification trees to solve the problem at hand. Decision or Classification tree is the most basic machine learning technique. It is generally the easiest to implement, relatively easy to understand and debug, easy to customize and also the generally the fastest in terms of learning and classification. Although, the speed and simplicity generally leads to a compromise on the accuracy of the prediction, I was confident that a well-tested and customized tree learning system would perform better than a neural network or a SVM as the tree would yield faster results.

A decision tree is a tool that represents an algorithm in the form a graph or binary, tertiary or n-ary tree with each node and branch having a certain associated outcome, weight in terms of outcome, and probability.

In machine learning, a decision tree can be seen as a predictive model which makes decisions on a branching series of Boolean tests. In computer logic terms, it can be viewed as a series of nested “if-elses”.

Figure 1 : Sample instances data and Decision Tree

Above is a sample decision tree. Here, the variable that we are trying to predict or the “target variable” is the “PLAY” variable. Given a set instances each with different values for OUTLOOK, HUMIDITY and WINDY and the resulting value for PLAY, a tree is constructed as above. Now, when given an instance with values for OUTLOOK, HUMIDITY and WINDY, the machine learning system should be able to predict the value for the PLAY variable by traversing through the tree, based on the conditions in the instance given.

The above example is a simplified version of the decision trees that are generally built and used as the data is much more complex.

Choosing the root of the tree, or the top node is done usually by measuring the entropy (or its inverse- information gain) for each attribute. Entropy is given by the formula:

$$\bigcirc \text{Entropy}(S) = \sum_{(i=1 \text{ to } l)} -|S_i|/|S| * \log_2(|S_i|/|S|)$$

- S = set of examples
- S_i = subset of S with value v_i under the target attribute
- l = size of the range of the target attribute

What this means is that the node which gives the least entropy and the most information gain will be chosen as the root node. All subsequent decisions of splitting the tree are also based on the entropy values.

Most modern decision trees are based on the C4.5 algorithm. The algorithm states:

- Choose the ***best* attribute(s) i.e. (the attribute with the most information gain value) to split** the tree on this attribute and make it a decision node.
- **Repeat this process for recursively for each child**
- **Stop when either:**
 - All the instances have the same target attribute value
 - There are no more attributes
 - There are no more instances.

2.3 Overfitting, Pruning and Cross Validation

After training on a sufficient amount of data, the model is assumed to have reached a point such that it can, on the basis of its inductive bias, predict the outcomes for data which it has previously not encountered. However, if the training was too long or if the learning examples are very rare occurrences, the trained tree model may create branches for specific features that do not make a difference to the target feature. In such a case, the performance of the model fitting the training data would keep on increasing, however, the performance of predicting unknown will decrease. This phenomenon is known as “overfitting”.

Overfitting can be understood easily by looking at the training data as divided into two parts: data that provides information about the target feature, and data that does not, which is considered as noise. Hence, noise needs to be removed or ignored from the training set. One of the ways of doing it is “cross-validation” followed by “pruning” the tree.

In cross-validation, the training data is partitioned into two sets, called the training set and the validation set. The system is trained on the training set and then predictions are made by the system on the validation set. The algorithm then compares the predicted values to the actual known values in the validation set, and provides statistics, most important of which is the “root mean squared error rate”. Based on the root mean squared error rate, the system decides the pruning strategy for the tree. So as to avoid any information bias, several rounds of cross validation are performed, with different subsets

of the training data to provide the best possible measure of the error rate. 10-fold cross validation is the most commonly used strategy.

The predictive accuracy of a decision tree depends a lot on its size. A very large tree may tend to overfit the data and provide bad prediction accuracy on generalized data. A small tree on the other hand, may not have captured certain important features of the training data. Pruning a decision tree, involves removing nodes, or making sub trees into leaf nodes. The most recommended strategy in decision tree building is to let the tree grow until each node has at least a few instances and then prune all those nodes which do not provide newer information.

2.4 Classifying an unseen instance

Before the tree is built, entropy is calculated for each feature. In this process, the minimum and maximum values for a particular feature become known to the system. The decision tree is nothing but a set of rules.

When the system is asked to classify an instance, it simply goes through the rules, as it would if it was a computer program with “if-else” conditional statements. The decision is made according to whichever conditions are satisfied by the instance. If none of the rules are satisfied, the decision tree has a final rule, which says that it should default to a particular value for the target variable. In the above example, the default value is considered to be “PLAY” as it has more occurrences than “DON’T PLAY”. Similarly, in the data provided by IBM, the default value would be “FALSE”.

2.5 Machine Learning environments and libraries

WEKA stands for Waikato Environment for Knowledge Analysis. It is an open source project in Java for machine learning. It was developed at the University of Waikato, New Zealand. WEKA is an open source project, and hence contains a large number of libraries and implementations of various machine learning algorithms.

MATLAB (MATrix LABoratory) is software for numerical computing and statistical analysis and also an integrated development environment which uses its own programming language. MATLAB primarily works in number matrices and has provisions for visualization of functions and data, libraries which implement various algorithms, and interfacing with programs written in other languages, etc. MATLAB has various different suites developed specifically for machine learning. As such, MATLAB is the most reliable and robust machine learning suite available with the most convenient documentation and standards.

Other machine learning environments include RapidMiner, KNIME, Scilab and Octave (open source alternatives to MATLAB). Each library or software suite has its own pros and cons. The idea was to find a solution such that the accuracy can be maximized through repeated trials and modifications. Hence it was also necessary that the software chosen would be required to carry out training and classification on instances in the order of millions relatively quickly.

3. Phase 1 – Using MATLAB for the IBM Challenge

MATLAB is one of the most powerful statistical software tools available. It allows users to install specific sets of tools and libraries, such as the neural networks or SVN toolboxes, and the machine learning suite which was used. The toolboxes provide functions which can be then referenced directly in your code.

Furthermore, the comma separated format that the data was provided in was ideal for MATLAB to directly work without having to modify the data much. MATLAB also provided results for training and classification relatively much quicker than other libraries or tools. Since there was limited time in the challenge, MATLAB was ideal in quickly setting up a system and testing the results.

3.1 Setting up and using MATLAB

Correctly identify the operating system, version of MATLAB to download and type of license and download the MATLAB installation file from the MATLAB software downloads page.

In order to start using MATLAB, you would need to have some basic knowledge of matrices, and how to load data into MATLAB.

The data provided by IBM was a single 3 GB comma separated value (“.csv”) file. There was no need to transform the data in any way, as MATLAB is capable of reading in “.csv” files in their original format.

However this file needed to be broken down into several smaller files as MATLAB was unable to load the entire file by itself. The data was loaded using MATLAB's "Import Data" tool. Once the file was loaded, the required columns were selected in the Import Data tool, and MATLAB converted the large csv file into a ".mat" file. The ".mat" file was about 10 times smaller than the original ".csv". As a result, it was possible to load all the parts of the data into MATLAB, and then concatenate the ".mat" files within the program code. Here, the data was automatically loaded as a number matrix, and was available in the MATLAB workspace in the form of a matrix file (".mat").

The next step was to consolidate all the matrices and then start our operations on the consolidated matrix. A similar procedure of breaking down and consolidating was required for the evaluation dataset (unlabeled data) as well. Some of the MATLAB code is shown below.

3.2 Preview of the MATLAB code

```
% Author : Chinmay Bhawe
% Applying Classification Trees to solve the IBM TGMCSJSU Pilot Challenge
% problem
% Basic Classification Tree with optimal Pruning.
% To Enhance : Ensemble learning method using many small classification
trees

%%Concatenate input matrix from .mat files
% After concatenating the data into .mat files, preprocess the data to get
% it into the matrix format that we require.
load('importedtrainingdata.mat')
LearningMatrixSingle=[learningMatrix;part9;part10;part11;part12;part13;part14
;learningMatrix2;learningMatrix3;learningMatrix4;learningMatrix5];
clearvars -except LearningMatrixSingle

%convert to single precision and remove elements from workspace
LearningMatrixSingle=single(LearningMatrixSingle);
%clearvars -except LearningMatrixSingle

%%Form input matrices for Classification Tree and build Classification Tree
%get everything in proper form for input to classification tree
InputResults=LearningMatrixSingle(:,343);
LearningMatrixSingle=LearningMatrixSingle(:,2:342);

LearningTree = ClassificationTree.fit(LearningMatrixSingle,InputResults);

clearvars LearningMatrixSingle InputResults

%prune tree
PrunedLearningTree=prune(LearningTree,'level',200);
%ideal pruning level from experiments on smaller data sets

%%Get the data to predict in the proper matrix form needed
load('data_to_predict.mat');
matrix_to_predict=single(matrix_to_predict);
matrix_to_predict=matrix_to_predict(:,2:342);

%Put the pruned Classification Tree to work
Prediction = predict(PrunedLearningTree,matrix_to_predict);

%write the results to csv file
csvwrite('Chinmay_submission.csv',Prediction);
```

Figure 2: Preview of the MATLAB Decision Tree program code

3.3 How to run the code

(A) To run prediction for IBM evaluation dataset

1.) Make sure the following ".mat" files are present in your local system's MATLAB Folder, usually located at: "C:\Users\USER\Documents\MATLAB"

- importedtrainingdata.mat

- data_to_predict.mat

2.) Also copy the following ".m" file to the MATLAB folder

- ChinmayClassificationIBM.m

3.) In MATLAB, open ChinmayClassificationIBM.m, and just run it.

4.) It will take anywhere between 2 hours to 5 hours depending on your system's configuration. The output will be a "Chinmay_submission.csv" file created in the MATLAB folder.

(B) To run prediction for accuracy test

1.) Make sure the following ".mat" files are present in your local system's MATLAB Folder, usually located at: "C:\Users\USER\Documents\MATLAB"

- importedtrainingdata.mat

2.) Also copy the following ".m" file to the MATLAB folder

- ChinmayClassificationAccuracyTest.m

3.) In MATLAB, open ChinmayClassificationAccuracyTest.m, and just run it.

4.) It will take anywhere between 1 hour to 3 hour depending on your system's configuration. The output will be a "Chinmay_submission_accuracy_test.csv" file created in the MATLAB folder.

5.) Also, the MATLAB console will show the statistics for accuracy (number of wrong predictions, and percentage of wrongly predicted rows).

MATLAB also provides metrics about the programs performance such as time required, etc.

Figure 3: Preview of results and run statistics

3.4 Results for Phase 1

Using MATLAB, various tests were performed using permutations and combinations of cross validation folds and tree pruning. A model was built and the tested on an unlabeled version of the training data set with a prediction accuracy of nearly 99%.

It was noticed that training the system on the entire data set would take upwards of 3 hours. Since I wanted to include a little bit of pre-processing, cross validation and also tree pruning,

Consequently, the unlabeled data set was classified using the model, and the results were provided to IBM.

4. Phase 2 – Using Weka for Cloud based system

The second phase of the project involved the migration of the machine learning system built in phase 1, on to a cloud server. Amazon Web Services (AWS) was the chosen cloud server provider as it was the most popular and also provided the largest choice of servers, operating systems, and services.

Unfortunately, AWS does not allow paid software to be hosted on its servers, as is the case with most cloud service providers. Hence, it was necessary to find an open source library or machine learning software which could help solve the same problem of classification of a very large data set, and can be hosted on a cloud server.

4.1 Why Weka?

Weka is a machine learning software suite developed in Java. It provides facilities for all the steps involved in solving a machine learning problem- data conversion, pre-processing techniques, classification, categorization and visualization. Weka commands can be carried out via the command line.

Figure 4: Weka Home GUI

It also provides a GUI on supporting systems, which makes it very easy in terms of understanding the flow of data, and visualizing the results. However, the GUI component does not work well when dealing with very large data, as was the case in this project.

Figure 5: Weka file pre-process example

On the other hand, the Weka Simple Command Line Interface (CLI) is lightweight in terms of memory, and provides much more scope for dealing with very large files.

Figure 6: Weka file classification example

Out of all the other machine learning libraries, Weka was the most cited and recommended. It was the most well documented, and also since it was written in Java, understanding and customization became more convenient.

4.2 Setting up the AWS Server

The first step is to set up an Amazon Web Services account. This is freely available and requires only a valid email id and a method of payment such a credit card. Once logged, in, navigate to the AWS Management Console and then to the EC2 (Elastic Cloud Compute) module.

Figure 7: AWS Launch EC2 Instance

Here click on the “Launch Instance” button, follow the steps to set up a server. Choose an operating system, and the type of server required (there are several predefined levels that you may choose from). The server used for this experiment was a “M1.XLarge” server instance with Ubuntu 12.10 OS.

Figure 8: AWS Choose Operating System

The server used for this experiment was a “M1.XLarge” server instance with Ubuntu 12.10 OS.

When setting up the server, AWS will provide a “.pem” file, which will provide authentication for logging in to your server. You can access the command line of the server via Amazon’s own SSH client, or via popular SSH clients such as “Putty”.

4.3 Installing and setting up Weka

The installation steps for Weka can be found on the Weka website at [12].

The steps are dependent on the operating system on which Weka needs to be installed.

On a Windows or Linux machine, the procedure is to download the zip/tar archive file, extract the package to the required folder and add the path to the “CLASSPATH” variable.

On Ubuntu systems, the process is as simple as just issuing the command “apt-get install Weka” on the command line and then adding the “weka.jar” file to the CLASSPATH as is demonstrated in the next section.

4.4 Steps to run training and classification using Weka

After setting up the server and installing Weka, the data needs to be uploaded to the server. This can be done using simple file transfer tools such as FileZilla or WinSCP.

Figure 9: WinSCP File Transfer demo

Running the training and classification consists of basically four steps – Data conversion to Weka format, running the training command and saving the model built, running the classification, and then running the java program to determine the accuracy.

For using Weka, unlike MATLAB, there was quite a lot of data manipulation work required before the data was in the correct format for Weka to ingest. Weka provides a tool to convert csv files to the “attribute-relation file format” or “.arff”. A sample “.arff” file is also given below. The ARFF format divides the file into 2 sections, the first section contains information about the relation and the different attributes (features) and data type. The second section contains the data in a comma separated format.

1.) **Converting the CSV data into ARFF format.**

Weka provides a function to convert comma separated value format files into the “arff” format which is readable by Weka. The below command requires that the

original “csv” file is free of missing values, errors, and extra spaces. I made use a various different text editing tools such as “notepad++”, “010 Editor” and reCSVEditor” when troubleshooting the arff convertor.

Command : `$> java weka.core.converters.CSVLoader`
`/pathtofile/filename.csv > filename.arff`

Figure 10: Sample ".arff" file

2.) If on a Linux system, **switch to root** user by:

Command : `$> sudo su`

3.) If on a Linux system, **export the “weka.jar” file to the CLASSPATH**. Use this command to tell the JVM where it can find the Weka jar file by issuing the

following command:

Command :`$> export`

`CLASSPATH=$CLASSPATH:/path_to_weka_file/weka.jar`

Example: `$> export`

`CLASSPATH=$CLASSPATH:/usr/share/java/weka.jar`

If unsure of where the “weka.jar” file is located, run the following command :

`$> find / -name weka.jar`

Output Example: `/home/ec2-user/data/weka-3-7-9/`

The output of this should be entered in the “/path_to_weka_file” part of the export command.

- 4.) **Training the system.** This is done by the telling Weka to build a classification tree using Weka commands and then save the model in a “.model” file.

Command : `$> java -Xmx8g weka.classifiers.trees.J48 -t`

`/path_to_training_file/trainingfile.arff -d`

`/path_where_to_save_model_file/trained.model`

Example: `$> java -Xmx8g weka.classifiers.trees.J48 -t`

`/home/ec2-user/data/trainingfile.arff -d /home/ec2-`

user/data/trained.model

```
root@ip-10-68-73-8: /home/ubuntu

System information as of Wed May 1 05:06:49 UTC 2013

System load: 0.01          Processes:           83
Usage of /: 2.3% of 492.15GB Users logged in:      0
Memory usage: 0%          IP address for eth0: 10.68.73.8
Swap usage: 0%

Graph this data and manage this system at https://landscape.canonical.com/

150 packages can be updated.
66 updates are security updates.

Get cloud support with Ubuntu Advantage Cloud Guest
http://www.ubuntu.com/business/services/cloud

Use Juju to deploy your cloud instances and workloads.
https://juju.ubuntu.com/#cloud-precise
*** /dev/xvda1 will be checked for errors at next reboot ***

Last login: Sat Apr 27 04:38:20 2013 from c-50-148-160-137.hsd1.ca.comcast.net
ubuntu@ip-10-68-73-8:~$ sudo su
root@ip-10-68-73-8:/home/ubuntu# nohup java -Xmx10g weka.classifiers.trees.J48 -t /home/
e/ubuntu/dataforweka/trainingdatafull.arff -d /home/ubuntu/dataforweka/final.model &
```

Figure 11: J48 Tree training command example

5.)

Here, the “-Xmx8g” tells the java virtual machine that it can have a maximum heap memory size of 8 gigabytes. Similarly, “-Xmx1g” would limit the java heap memory to 1 gigabyte, -Xmx500m to 500 megabytes, and so on.

If there is no “-Xmx” option used, it will default to the default java heap memory.

This is **necessary to use when** we want to train/classify using really large files (such as the IBM data file which is 3 gigabytes in size) as it makes sure that the Weka process does not run out of memory and throw an exception. Also, it is

advisable the use the Linux command “nohup” when working with large data sets as it will keep the process running in the background, regardless of whether the connection to the server is alive or not. Often, training large files takes several hours. **Note the “&” at the end of the command.**

Example: `$> nohup java -Xmx8g weka.classifiers.trees.J48 -t /home/ec2-user/data/trainingfile.arff -d /home/ec2-user/data/trained.model &`

This is **not necessary to use** if we are working with small files.

The “-t” specifies that the file is for training. The “-d” specifies Weka to dump the tree output into the model file.

```
root@ip-10-68-73-8: /home/ubuntu/dataforweka
J48 pruned tree
-----
outlook = sunny
|  humidity <= 75: yes (2.0)
|  humidity > 75: no (3.0)
outlook = overcast: yes (4.0)
outlook = rainy
|  windy = TRUE: no (2.0)
|  windy = FALSE: yes (3.0)

Number of Leaves   :    5
Size of the tree   :    8

Time taken to build model: 0.02 seconds
Time taken to test model on training data: 0 seconds

=== Error on training data ===
Correctly Classified Instances      14          100   %
Incorrectly Classified Instances     0           0   %
Kappa statistic                     1
Mean absolute error                  0
```

Figure 12: J48 tree building statistics

6.) Using the model to **classify the test data**. This is done by the following:

```
Command : $>java -Xmx8g weka.classifiers.trees.J48 -l  
/path_to_model_file/trained.model -T  
/path_to_test_file/test.arff -p 0 >  
/path_to_save_results_file/results.txt
```

```
Example: java -Xmx8g weka.classifiers.trees.J48 -l  
/home/ec2-user/data/trained.model -T /home/ec2-  
user/data/test.arff -p 0 > /home/ec2-  
user/data/results.txt
```

Here, the “-l” specifies the model to load, “-T” specifies that the arff file is for classification.

The “-p 0” specifies to output the classification result only. The output is the redirected using the “>” operator to a text file. If the redirection is not done, the output is visible on the console.

There are many other options which may be used with the Weka commands and

these can be found at [6].

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

=== Predictions on test data ===

inst#    actual    predicted    error    prediction
1        2:no      2:no        1
2        2:no      2:no        1
3        1:yes     1:yes       1
4        1:yes     1:yes       1
5        1:yes     1:yes       1
6        2:no      2:no        1
7        1:yes     1:yes       1
8        2:no      2:no        1
9        1:yes     1:yes       1
10       1:yes     1:yes       1
11       1:yes     1:yes       1
12       1:yes     1:yes       1
13       1:yes     1:yes       1
14       2:no      2:no        1
```

Figure 13: Sample results file after classification

7.) **Copy the results file to local machine.**

8.) **Testing the accuracy of the prediction.** This is done by running a simple java program “*AccuracyCheckForClassifiedFile.java*”. This can be done via command line or in an IDE such as Eclipse or Netbeans. The program when run, will ask for the path to the results file. Once the path is entered, the program will output the statistics.

Example:

Please enter the path to the results file(without any spaces):

Example: `C://Test_folder/resultsclassified.txt`

No of WRONGLY classified instances: 751.0
No of CORRECTLY classified instances: 354667.0
Total number of Instances : 355418.0
Prediction Accuracy :99.788704 %

4.5 Results for Phase 2

Several experiments were conducted using different cross-validation levels and pruning strategies. The aim was not only to increase prediction accuracy, but also to test what effect preprocessing and pruning techniques have on training time. The general training time for the entire data set of 3 gigabytes was about 10 hours. Below is a table of the experiments conducted, and their respective results:

Test number	Size of training data	Time taken to train	Cross Validation	Size of testing data	Time taken to classify	Prediction accuracy
		approx.			approx.	
1	1 MB	3.3 seconds	10 fold	1 KB	< 1 second	99.81%
2	100 MB	45 minutes	10 fold	1 MB	< 1 second	99.82%
3	500 MB	1.5 hours	10 fold	10 MB	4 seconds	99.85%
4	1 GB	3 hours	10 fold	200 MB	32 seconds	99.65%
5	2 GB	4.5 hours	10 fold	200 MB	30 seconds	99.91%
6	2 GB	4.5 hours	10 fold	500 MB	45 seconds	99.81%
7	2 GB	4.5 hours	10 fold	1 GB	1 minute	99.83%
8	2 GB	5.5 hours	20 fold	500 MB	45 seconds	99.85%
9	2 GB	5.5 hours	20 fold	1 GB	1 minute	99.84%
10	3 GB	9.5 hours	10 fold	10 MB	5 seconds	99.85%
11	3 GB	9.5 hours	10 fold	200 MB	40 seconds	99.84%
12	3 GB	11.5 hours	20 fold	500 MB	1.5 minutes	99.87%

13	3 GB	12 hours	20 fold	1 GB	1.5 minutes	99.86%
14	3 GB	9.5 hours	10 fold	1 GB	1.5 minutes	99.82%
15	3 GB	9.5 hours	10 fold	300 MB	45 seconds	99.83%

Figure 14: Table of experiments in Weka

- As it can be seen from the table, the prediction accuracy was always above 99.5% so long as the size of the training set was much larger than that of the testing set.
- The training time was also very large, compared to the classification time. This confirms the fact that decision trees are fast predictors but generally slow learners.
- All experiments were carried out on an AWS M1.xLarge EC2 server (64 bit platform, 15 Gigabytes of memory, 4 core processor)
- Experiments carried out with smaller, less powerful server instances proved to be failures as either the command ran out of memory when loading the data file, or it took above 24 hours for the process to complete. Since there is no way to determine the progress of the java process, the processes were stopped after if no result showed up even after 24 hours of running.
- The training command for the 3 GB data set required at least 3.5 hours using MATLAB and at least 9 hours using Weka.
- The time taken to classify is almost negligible as compared to the time taken to train. Even a gigabyte can be classified in less than 2 minutes.

4.6 Observations and Inferences

Figure 15: Training time vs. training data size

- As a general trend, the training time of the tree depends not only on the size of the data, but also on the number of folds of cross validation.
- The higher the number of cross validation folds, the more the training time.
- Training time also increases almost exponentially with respect to the size of the training data.
- The prediction rate increases very slightly from 99.81 to about 99.86 as we increase the cross validation folds, and as a result, the training time.
- The almost exponential increase in the training time is largely because of the number of features in the training set. The algorithm has to go through all the instances for each feature, in order to calculate the entropy. Hence, the training

time can be said to be largely proportional to the number of features in the training data, and the number of instances as well.

Figure 16: Time taken to classify vs. validation data size

- The above graph shows the time taken to classify in seconds versus the size of the evaluation data set. As can be seen from the graph, the classification time increase linear with increasing data size.
- The classification time being linear is logical as well. The algorithm has to go through all the instances only once, and make a decision at the end of each step. Hence, it does not matter how many features the data is made up of. The only thing that would matter is the number of instances.
- A general best practice, would be to have the training data set at least 5 times larger than the test data set. The number of cross validations should be chosen according to the prediction accuracy results, but 10 –fold is generally suggested.

- As stated before, there is no common best solution for all decision tree problems. The amount of cross validation, pre-processing, feature selection, filtering, etc. depends entirely on the kind of data that you are dealing with.

4.7 MATLAB vs. Weka

This project required both these libraries for the same purpose, but for different reasons. As can be seen from the classification results, the predictive accuracy of both the libraries is comparable. While MATLAB could train on the entire 3 gigabyte training set in less than 4 hours, it took Weka an average of 9 and half hours. In both the cases, the classification step was very fast, and took at most a couple of minutes to classify the unlabeled IBM data. Weka provided a lot of features which MATLAB did not, such as variations in cross-validation, different pre-processing techniques such as SMOTE filters, and also provided visualization for the tree and the output. MATLAB had the best documentation, and was easy to learn. Weka on the other hand, involved a lot of trial exercises to understand the command line interface. However, Weka being an open – source technology, it could be hosted on a cloud server.

In short, MATLAB was more powerful and provided faster results and served the purpose well for the IBM challenge, and Weka on the other hand, being equally competent more than satisfied the requirements of the second phase of the project.

5. Conclusion

Both the phases in this project were completed successfully. MATLAB was used to successfully carry out tests on the very large data set, and classification of the evaluation data set was done using the best model achieved through the experiments. In phase 2, Weka was the chosen machine learning library for hosting the system on a cloud server. Powerful Ubuntu and Linux machines were set up using Amazon Web Services and experiments were carried out using J48 decision tree, a java based implementation of the C4.5 decision tree algorithm, in Weka. Model training was successfully carried out on the big data set given by IBM and classification was carried out using the most optimal parameters found from the tests.

In both phases, a prediction accuracy of over 99% was achieved. The maximum training time was about 11 hours and the minimum was 3 hours for the very large data set. Thus the objectives were met and the project was a success.

6. Proposed future work

Now that we know that such very large data sets can be trained on and classified on the cloud, future work on this project can be to try and solve a real world problem using this set up. One example would be to set up a system for matching candidate resumes, to job postings using machine learning. After converting resumes, and job listings into a feature vector form, the machine learning system developed in this project will be trained on a large number of jobs listing data. The when a user wants to search for a job matching his or her skills and resume, it is simply entered into the system and the decision tree will be able to predict the closeness, or relevance of each job posting to the resume. This can be then compared to a job search using the traditional technique of keyword matching.

Furthermore, this machine learning system on the cloud can be converted into a web service, such that the data conversion, training and classification commands can be carried out via a web interface, without the user actually having access to the server itself.

There can be many more applications of this system for other real world classification problems.

7. References

- [1] G. Appenzeller, I. Keslassy and N. McKeown, “Binary Classification of text”, 2005
- [2] Statistical classification. (2012, September 25). In *Wikipedia, The Free Encyclopedia*. Retrieved 02:40, December 4, 2012, from http://en.wikipedia.org/w/index.php?title=Statistical_classification&oldid=514565024
- [3] WEKA. (2011, May 4). In *Wikipedia, The Free Encyclopedia*. Retrieved 02:42, December 4, 2012, from <http://en.wikipedia.org/w/index.php?title=WEKA&oldid=427359442>
- [4] Sally Floyd and Van Jacobson, “Random Early Detection for Congestion Avoidance”, IEEE/ACM Transactions on Networking, 1993
- [5] Windows Installation for MATLAB R2011b (Student) Retrieved 21:12, February 6, 2013, from <http://www.usc.edu/its/matlab/windows.html>
- [6] “Running from the command line”, Weka Decision tree command line options, Retrieved 11:05, May 1 , 2013, from

http://www.cs.waikato.ac.nz/~remco/weka_bn/node13.html

[7] Overfitting. (2011, May 4). In *Wikipedia, The Free Encyclopedia*. Retrieved 17:42, May 1, 2013, from

<http://en.wikipedia.org/wiki/Overfitting>

[8] Pruning (decision trees). (2013, March 2). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:15, May 2, 2013, from

[http://en.wikipedia.org/w/index.php?title=Pruning_\(decision_trees\)&oldid=541684233](http://en.wikipedia.org/w/index.php?title=Pruning_(decision_trees)&oldid=541684233)

[9] IBM Watson - How to build your own "Watson Jr." in your basement,

Retrieved 00:15, May 2, 2013 from

https://www.ibm.com/developerworks/mydeveloperworks/blogs/InsideSystemStorage/entry/ibm_watson_how_to_build_your_own_watson_jr_in_your_basement7?lang=en

[10] MATLAB classification and regression trees, Retrieved 00:15, May 2, 2013 from

<http://www.mathworks.com/help/stats/classification-trees-and-regression-trees.html>

[11] Amazon AWS EC2, Retrieved 00:16, May 2, 2013 from

<http://aws.amazon.com/ec2/>

[12] Downloading and installing Weka, Retrieved 00:18, May 2, 2013 from

<http://www.cs.waikato.ac.nz/ml/weka/downloading.html>

[13] WinSCP Client, Retrieved 00:15, May 2, 2013 from

<http://winscp.net/eng/index.php>

[14] 010 Editor, Retrieved 00:22, May 2, 2013 from

<http://www.sweetscape.com/010editor/>

[15] Shams,Rusdhi, Weka Tutorial videos playlist, Oct 6, 2006 Retrieved 00:25, May 2, 2013 from

<http://www.youtube.com/user/rushdishams>